J Supercomput
DOI 10.1007/s11227-012-0866-7

# A proximity-aware load balancing in peer-to-peer-based volunteer computing systems

**Toktam Ghafarian · Hossein Deldari · Bahman Javadi · Rajkumar Buyya**



**Abstract** One of the main challenges in peer-to-peer-based volunteer computing systems is an efficient resource discovery algorithm. Load balancing is a part of resource discovery algorithm and aims to minimize the overall response time of the system. This paper introduces an analytical model based on distributed parallel queues to optimize the average response time of the system in a distributed manner. The proposed resource discovery algorithm consists of two phases. In the first phase, it selects peers in a load-balanced manner based on QoS constraints of request. In the second phase, a proximity-aware feature is applied to select the peer with minimum communication overhead among selected peers in the first phase. Two dispatching strategies are proposed for the load balancing based on stochastic analysis of routing in the distributed parallel queues. These policies adopt probabilistic and deterministic sequences to redirect requests to the capable peers in the system. Simulation results show that the proposed resource discovery algorithm improves the response time of user's requests by a factor of 1.8 under a moderate load.

T. Ghafarian (✉) · H. Deldari
Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
e-mail: ghafarian@stu-mail.um.ac.ir

H. Deldari
e-mail: hd@ferdowsi.um.ac.ir

T. Ghafarian
e-mail: tghafarian@yahoo.com

R. Buyya
Cloud Computing and Distributed Systems Laboratory, Department of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
e-mail: rbuyya@unimelb.edu.au

B. Javadi
School of Computing, Engineering and Mathematics, University of Western Sydney, Sydney, Australia
e-mail: b.javadi@uws.edu.au

Published online: 24 January 2013

**Keywords** Peer-to-peer computing · Volunteer computing · Resource discovery · Load balancing · Distributed parallel queue · Proximity-aware scheduling

## 1 Introduction

Volunteer computing (VC) which benefits from idle cycles of desktop computers is an attractive cost-efficient platform for running scientific projects with heavy computation requirements [1–4]. Some of popular volunteer computing systems are BOINC [5], condor-like grid system [6–8], Entropia [9], XtremeWeb [10], Aneka [11], SZTAKI [12], QADPZ [13], and IPOP/WOW [14]. Peer-to-Peer (P2P)-based VC systems represent a decentralized, self-organized, and scalable environment for running applications such as PastryGrid [15], BonjourGrid [16], ShareGrid [17], CondorFlock P2P [18], and Self-Gridron [19].

Resource discovery algorithm has a great impact on overall performance of these systems. One of the main challenges for designing an efficient resource discovery algorithm is the load balancing policy. The objective function of load balancing is minimizing the overall response time of the system.

The main contribution of this work is to propose a proximity-aware load balancing strategy in the resource discovery algorithm of P2P-based VC systems. In our previous work [20] a distributed proximity-aware architecture for resource discovery in P2P-based VC systems was proposed. This architecture is named CycloidGrid, it distributes an incoming load among peers based on communication overhead and current load of peers. In CycloidGrid, we have shown that if we consider communication overhead among peers in the resource discovery algorithm, the average response time of the system decreases. In this research we focus on minimizing average response time and decreasing the overhead of resource discovery algorithm by stochastic analysis of routing in distributed parallel queues. The proposed policies are knowledge-free (i.e. they are not dependent on current load of each peer). Thus, they do not impose any overhead on the system. Also, deadline is added to the QoS constraints of BoT requests.

The proposed resource discovery algorithm consists of two parts. In the first part, a number of peers are selected fairly by one of the dispatching policies based on stochastic analysis of routing in the distributed parallel queues. The dispatching strategies take into account QoS constraints of request such as CPU speed and RAM or disk space requirements. In the second part, the proposed resource discovery algorithm decreases the communication overhead by selecting a peer with minimum communication delay among the advertised peers in the first part. Millions of heterogeneous resources are disseminated across geographically distributed peers in the P2P-based volunteer computing systems; therefore, running a job on a node with lower communication overhead can reduce the communication delay, and increase the overall performance. In summary our paper includes the following contributions:

- Providing an analytical queuing model for load balancing in P2P-based volunteer computing systems based on parallel non-observable queues;
- adapting the proposed analytical model for distributed resource discovery policy;

- proposing a probabilistic and deterministic dispatch policy for load balancing in the system to meet the QoS requirements of each request;
- evaluating the proposed policies under realistic workload models and different number of peers to show scalability of the system.

The rest of this paper is organized as follows. Section 2 presents a literature review. Section 3 discusses CycloidGrid environment including architecture and the resource discovery policy. Section 4 presents analytical queuing model for load balancing in P2P-based VC systems. This analytical model is based on routing in parallel queues. The proximity-aware load balancing policy is presented in Sect. 5. This section gives a detailed overview of applying analytical model for load balancing in the system. Section 6 describes the performance evaluation of the proposed policy under a realistic workload model. Conclusion and future directions are presented in Sect. 7.

## 2 Related work

There are several research works that have investigated load balancing and QoS constraints in the resource discovery algorithm of P2P-based volunteer computing systems. These researches can be divided into two categories: the first category is the load balancing based on information gathered from the peers on the system (knowledge-based approach). The second category uses analytical model for load balancing with the knowledge-free approaches, but these works have not considered QoS constraints. In the first category, we highlight the following work.

Kim et al. [21] proposed an approach for load balancing in the resource discovery algorithm of P2P-based desktop grid systems. The resource discovery algorithm is considered as routing problem in the CAN [22] space. CPU speed, memory, and disk space are considered as QoS constraints for each request. It searches a node whose coordinate in all dimensions satisfies or exceeds QoS constraints. The matchmaking algorithm distributes jobs among capable resources evenly based on aggregated load information along each dimension of the CAN overlay network. This method neglected proximity-aware feature.

Abdullah et al. [23] suggested a dynamic and self-organizing model for resource discovery in ad hoc grids. In this work, three types of agent named customer, producer, and matchmaker were introduced. The whole identifier space is divided into zones which has a dedicate matchmaker. The matchmaker uses a continuous double auction to perform resource allocation, and looks for matches among producers and consumers according to QoS requirements of the request. Required resource size, resource availability, deadline, and budget were studied as QoS constraints. The authors defined a mechanism to calculate the matchmaker workload (TCost) based on the number of request/offer messages to be processed in the ad hoc grid. TCost based on threshold is applied for dynamic segmentation and de-segmentation, and balancing a load among different matchmakers. Moreover, resource discovery algorithm ignores proximity of nodes.

Mastroianni et al. [24] proposed a super-peer-based resource discovery algorithm for P2P-based volunteer computing systems. Their resource discovery algorithm consists of two phases: job-assignment and data-download phase. In the job assignment,

a job manager generates a number of job's advert based on QoS constraints like characteristics of platform, and sends them to the local super-peer and some of other super-peers in the system. Workers generate a job query. Then, job query travels the network through the super peer interconnections until its time-to-live parameter decreases to zero or the job query finds a matching job's advert. In the data-download phase, the worker sends a data query, and downloads a data file from a closest data centre. In this work, load balancing is ignored.

Lazaro et al. [25] proposed a decentralized resource discovery algorithm that meets QoS constraints of request in P2P-based VC systems. The authors used common KBR overlay network, and the requested number of resources are considered as QoS constraints. Three main agents (worker, client, and matchmaker) were defined in the system. A worker sends advertisements to multiple matchmakers in the system. When a client needs resources, it asks matchmaker, and matchmaker searches among advertisements in order to find possible matches. In this work only QoS requirements of request is studied, but load balancing and proximity-aware feature are neglected.

Di et al. [26] presented a decentralized scheduling algorithm for dynamic load balancing in a self-organized desktop grid environment. A dynamic Newscast model [27] is used as unstructured P2P overlay. In this research, each peer gathers load information of its neighbors based on epidemic gossip protocol. The average load level on participating nodes is used to distinguish overloaded and under loaded nodes in the system. A node is in a load balanced state if its current load closes to average load level. If it is overloaded or under loaded, it is improved by migrating any process into it or out from it. An autonomous scheduler designed on each node performs process migration. The system decreases migration overhead by considering process workload and bandwidth between two relative nodes. QoS constraints of request are ignored in their work.

In the second category there are a few research works that use knowledge-free method for load balancing in the grid systems. Some of these works are as follows.

Di et al. [28] improved a previous work [26] to design a conflict-minimizing load balancing algorithm, which can balance uneven workload in dynamic P2P desktop grids. In this work, each heavy loaded node selects light loaded node for task migration based on a distributed Bernoulli probabilistic model. They argued that asynchronously selecting target light node by each heavy loaded node in the competitive circumstances could be regarded as a set of Bernoulli trials. By using the decentralized Bernoulli model, decision conflict of task migration is decreased, and the efficiency of load balancing method is improved. This work is a combination of knowledge-free and knowledge-based method. Because at first any peer gathers load information of its neighbors based on epidemic gossip protocol; then, it uses Bernoulli model to improve the performance of load balancing algorithm.

Chatrapati et al. [29] considered the grid system as $n$ heterogeneous computing resources connected by a communication network using $m$ users. Each node is modeled as an M/M/1 queuing system, and all jobs are supposed to have the same size. The communication overhead between two nodes is considered independent of the nodes, and computed by total traffic through the network. They used a competitive equilibrium solution for load balancing in computational grids. The competitive equilibrium problem of load balancing finds equilibrium prices for the computing resources; then,

**Table 1** Comparison among previous studies and proposed load balancing policy

| Studies | Platform | QoS constraints | Load balancing policy | Proximity-aware feature |
|---|---|---|---|---|
| Kim et al. [21] | CAN overlay network | CPU speed, memory, and disk space | Knowledge-based | No |
| Abdullah et al. [23] | Pastry overlay network | Required resource size, resource availability, deadline, and budget | Knowledge-based | No |
| Mastroianni et al. [24] | Super-peer overlay network | characteristics of platform | No | Relatively, just for downloading data file |
| Lazaro et al. [25] | common KBR overlay network | requested number of resources | No | No |
| Di et al. [26] | Unstructured P2P overlay network based on Newscast model | No | Knowledge-based | Relatively, just for load migration |
| Di et al. [28] | Unstructured P2P overlay network based on Newscast model | No | Combination of knowledge-based and knowledge-free method | No |
| Chatrapati et al. [29] | Centralized system | No | Knowledge-free based on M/M/1 queuing system | No |
| The proposed load balancing policy | Cycloid | CPU speed, memory/disk space, and deadline | Knowledge-free based on GI/GI/1 queuing system | Yes |

it specifies allocation of user jobs to the nodes at these prices such that each user optimizes objective function against budget constraints. In this work authors proposed a load balancing strategy based on knowledge-free method but in a centralized manner. Also, the resource discovery algorithm ignores QoS constraints of each job in the system.

Table 1 presents a comparison among these studies and the proposed load balancing policy in this paper, in terms of platform, QoS constraints, load balancing, and proximity-aware feature.

## 3 CycloidGrid environment

In this section, a brief overview of CycloidGrid architecture and resource discovery policy is provided. Interested readers can refer to [20] for more detail about CycloidGrid.

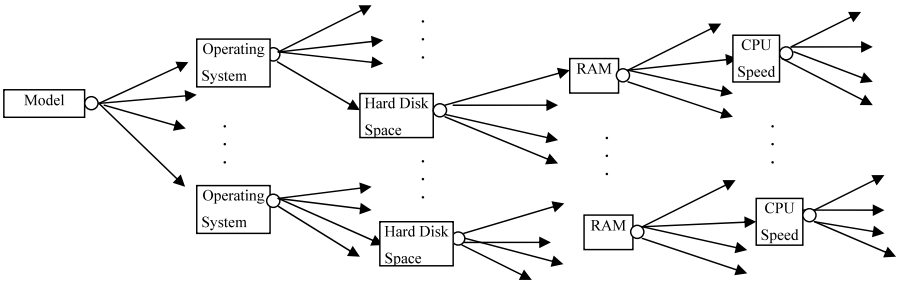


**Fig. 1** Decision tree for classification of resources based on their attributes

### 3.1 Resource and application models

Any volunteer resource in VC systems (e.g. desktop, laptop, tablet computers, smart phones, and servers) can be assumed as a resource in CycloidGrid [42]. These resources are heterogeneous, and have intermittent or permanent Internet connectivity [41]. Resource and peer are used interchangeably in this paper. Each job is considered to be a Bag of Tasks (BoT) application containing some of independent parallel tasks, which will be run on a single resource. Because some of resources in VC systems have less connectivity [41] (e.g. wireless connection); thus, many tasks are assigned at once to keep the resource busy until the next connection.

### 3.2 Architecture

CycloidGrid is a proximity-aware resource discovery architecture in P2P-based volunteer computing systems. It uses Cycloid [30] as a P2P overlay network. Cycloid is a constant-degree structured P2P overlay with $n = l.2^l$ nodes where $l$ is a dimension. All nodes are classified into some clusters. Each node is identified with a pair of indices $(k, a_{l-1}a_{l-2}\ldots a_0)$ where $a_{l-1}a_{l-2}\ldots a_0$ is a cubical index identifying its position among $2^l$ existing clusters, whereas $k$ is a cyclic index that identifies its position among $l$ nodes in its cluster.

Three types of node are defined in CycloidGrid. These nodes are called *reporting* node, *host* node, and *client* node. The reporting nodes are responsible for keeping resource attribute values of peers in the system. These attributes include model, operating system, CPU speed, RAM, and available hard disk. Host node can find suitable resource to run a job, when it receives a lookup request, and it can run its associated jobs. The client node sends a lookup request for running a job. It keeps executable code of the job, input and generated output files.

Decision tree (DT) is applied to classify resources based on resource attributes into some clusters, as it is shown in Fig. 1. Four attribute values are selected in each level of DT. Consequently, the number of clusters in DT is $4^5 = 1024$ clusters.

Each cluster of DT keeps the attribute values of subset of resources with identical operating system and processor model, and close CPU speed, RAM, and hard disk size. These clusters assign to the first 1024 clusters of CycloidGrid, and they are called reporting clusters. The remaining clusters of CycloidGrid are called host clusters. Consequently, we have two types of cluster in CycloidGrid: *reporting* clusters

and *host* clusters. Reporting clusters keep reporting nodes, and host clusters contain host/client nodes. Each reporting cluster contains three reporting nodes with similar resource attribute values. One of these reporting nodes is called primary reporting node that has the largest cyclic index in the corresponding cluster, and the other ones are called replica reporting node. Replica nodes have snapshot of resource information of primary node.

### 3.3 Resource discovery policy

Each request (job) is served within a single peer in CycloidGrid. It has the following characteristics:

- Number of independent tasks
- Estimation of each task duration
- QoS constraints in terms of minimum CPU speed, minimum RAM or disk space requirement, and deadline.

Figure 2 illustrates a scenario in which a resource is selected for running a request. At first, client node sends a lookup request for its job to the randomly active host node in the system (Step 1). The selected host node is called an *injection* node. The injection node acts as a scheduler for this request. This node has two queues such that one queue belongs to the lookup request, and another one belongs to the jobs that should be executed on this node. Each injection node uses decision tree to find reporting clusters can be useful to search according to the QoS constraints of this request. As it was mentioned earlier, every reporting cluster has one primary and two replica nodes with the same resource attribute values. In this phase, injection node selects a reporting node with minimum communication overhead among these three reporting nodes in each selected reporting cluster (Step 2). Communication overhead in this research is computed by a network model based on queuing theory discussed in Sect. 6.

Each reporting node searches among its resource attribute values to find a resource that satisfies QoS constraints (Step 3). It uses a load balancing policy that is explained

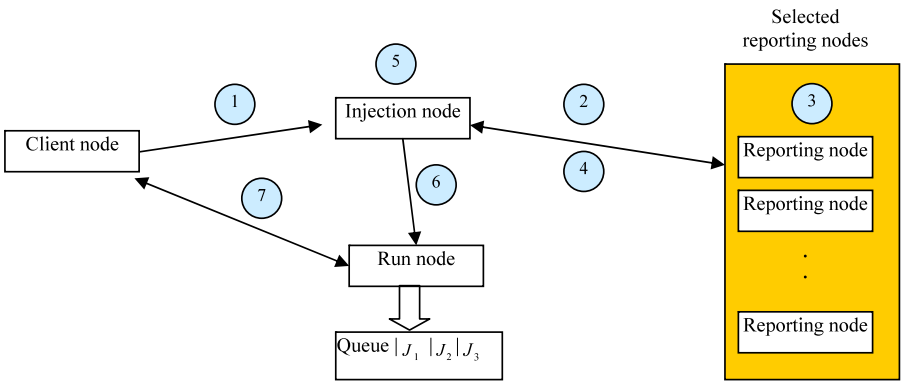


**Fig. 2** Resource discovery policy in CycloidGrid environment

in two following sections. Finally, the reporting node selects a resource among its resources, and sends the address of the selected resource to the injection node (Step 4). The injection node receives some resource offers for running its request from multiple reporting nodes. If the request does not have the deadline, the injection node will select a resource with minimum communication overhead to itself and the client node of this request. Whereas, if the request has deadline, the injection node will select a resource with higher priority to maximum CPU speed and lower priority to minimum communication delay. In order to optimize these two parameters, at first a resource with maximum CPU speed is selected; then, a resource with minimum communication delay is selected. If the selected resource in these two stages is identical, this resource will be selected. Otherwise, the resource with next minimum communication delay is selected until half of resources are chosen. If half of the resources are selected, and they are not identical with the resource having maximum CPU speed; then, the resource with next maximum CPU speed is selected. This process continues until these two resources are overlapped. The selected resource is called *run* node (Step 5). The injection node sends a job profile to the run node (Step 6). Finally, the run node puts this request in its queue, downloads the source code and input files of this request, and returns generated output files (Step 7).

## 4 Analytical queuing model

As we discussed in Sect. 3.3, each request is received by an injection node; then, it is redirected to subset of reporting nodes. Because each reporting node contains subset of resource attribute values, and it advertises suitable resource to run a request; therefore, the load balancing policy is considered in the reporting nodes of the system. The analytical model discussed in this section is applied in each reporting node to balance a load among its resource pool. The resource pool of each reporting node is divided into *logical clusters*. Each logical cluster contains subset of resources with close CPU speed. Thus, each reporting node is assumed to have a number of logical clusters. The objective function is to find the optimal arrival rate of incoming requests to each logical cluster a way that incoming requests are distributed evenly among the logical clusters in each reporting node. This section is followed by our proposed load balancing policy built upon the analytical model provided in this section.

The analytical model is based on routing in parallel queues. The queuing model that represents the whole system is shown in Fig. 3. In this model, it is assumed that the requests arrive into the system from all of client nodes with arrival rate $\lambda$ and variance $\sigma_I^2$. These requests are sent to the injection nodes; then, they are redirected to a subset of reporting nodes. These reporting nodes are selected by a decision tree based on QoS requirements. Therefore, each reporting node receives a subset of arrival requests in the system. This subset is estimated according to the following formula in each reporting node:

$$\hat{P}_i = \left\{ \begin{array}{ll} \dfrac{R_i}{\bar{R}} * \alpha & R_i < \bar{R} \\ \dfrac{R_i}{R_m} * \alpha & R_i \geq \bar{R} \end{array} \right\} \quad \begin{array}{l} 1 \leq i \leq k \\ 0 < \alpha < 1 \end{array} \tag{1}$$

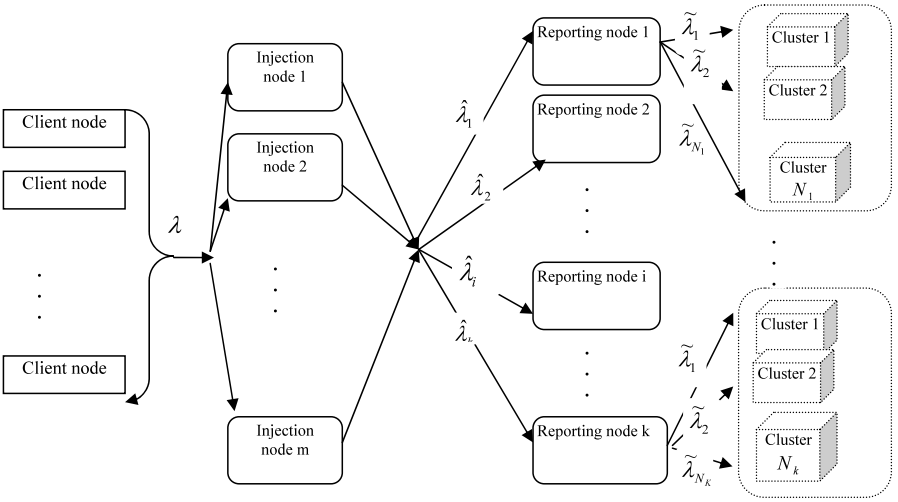


**Fig. 3** Queuing model for resource discovery in CycloidGrid

where $\hat{P}_i$ is a routing probability of incoming requests to reporting node $i$. $R_i$ is the number of resources in reporting node $i$, and $\bar{R}$, $R_m$ are average and maximum number of resources among all reporting nodes, respectively. $\alpha$ is a calibration factor between 0 and 1. For the sake of clarity, Table 2 gives the list of symbols that is used in this paper with their definitions.

Based on Fig. 3, reporting node $i$ receives incoming requests with arrival rate $\hat{\lambda}_i = \hat{P}_i \lambda$, and its variance can be computed by Wald's equation [32] as follows:

$$\sigma_{I_i}^2 = \frac{\sigma_I^2 \hat{P}_i + (\lambda)^{-2}(1 - \hat{P}_i)}{(\hat{P}_i)^2} \tag{2}$$

By having arrival rate and variance of incoming requests to each reporting node, we aim to find the optimal arrival rate of these requests to each logical cluster in order to balance the requests in the system. In this analytical model, each logical cluster (LC) can be considered as a server with the given service rate. Therefore, by assuming a logical cluster as a single queue, and the reporting node as a router that redirect incoming requests to the logical clusters, the problem can be considered as a routing in the distributed parallel queues [33, 34]. Each reporting node acts as router in front of a number of logical clusters as heterogeneous multi-server parallel queues. The objective function in each reporting node can be expressed as follows:

$$\min \hat{\lambda}_i \hat{T}_i = \min \sum_{j=1}^{N_i} \tilde{\lambda}_j E(\tilde{T}_j) \tag{3}$$

Equation (3) aims to find the optimal arrival rate $\tilde{\lambda}_j$ to each LC queue $j$ in the reporting node $i$. However, any reporting node with arrival rate $\hat{\lambda}_i$ routes incoming requests to the LC queues immediately after its arrival according to the routing probability $\tilde{P}_j$. It is supposed that there is no queue in the reporting node, and it is fast

**Table 2** Description of symbols used in the queuing model

| Symbol | Definition | Symbol | Definition |
|---|---|---|---|
| $\lambda$ | Arrival rate of jobs into the system | $C^2_{I_j}$ | The squared coefficient of variance for the arrival rate on cluster $j$ at any reporting node |
| $\sigma^2_I$ | Variance of arrival rate of jobs into the system | $C^2_{S_j}$ | The squared coefficient of variance for service time on cluster $j$ at any reporting node |
| $\hat{\lambda}_i$ | Arrival rate of jobs into a reporting node $i$ | $\tilde{T}_j$ | Average response time of incoming requests on cluster $j$ at any reporting node |
| $\sigma^2_{I_i}$ | Variance of arrival rate of jobs into a reporting node $i$ | $\hat{T}_i = \tilde{T}_j$ | Average response time of incoming requests at reporting node $i$ |
| $\hat{P}_i$ | Routing probability of arrival rate to reporting node $i$ | $N$ | Number of resources in the system |
| $N_i$ | Number of logical clusters in reporting node $i$ | $K$ | Number of reporting nodes in the system |
| $R_i$ | Number of resources in reporting node $i$ | $S_m$ | Maximum average of processing speed among all logical clusters at any reporting node |
| $\tilde{\lambda}_j$ | Arrival rate of jobs into logical cluster $j$ after load distribution at any reporting node | $\bar{S}_j$ | Average processing speed of cluster $j$ at any reporting node |
| $\tilde{P}_j$ | Routing probability of arrival rate of logical cluster $j$ in any reporting node | $\bar{S}_{BoT}$ | Average of BoT size |
| $\bar{x}_j$ | Average service time of jobs on cluster $j$ in any reporting node | $\bar{E}_{BoT}$ | Average of task execution time in BoT request |
| $S_j$ | Service time distribution of cluster $j$ at any reporting node | | |

enough to do that because each reporting node only find capable resources and it is not used for running a job. We model each LC with a single server queue. Therefore, in reporting node $i$ LC queue $j$ has the arrival rate $\tilde{\lambda}_j = \tilde{P}_j\hat{\lambda}_i$ and its variance can be computed by Wald's equation [32] as follows:

$$\sigma^2_{I_j} = \frac{\sigma^2_{I_i}\tilde{P}_j + (\hat{\lambda}_i)^{-2}(1-\tilde{P}_j)}{(\tilde{P}_j)^2} \tag{4}$$

Service time of LC queue $j$ follows a given distribution $S_j$ with mean $E[S_j] = \bar{x}_j$ and the coefficient of variance $C_{s_j} = \frac{\sigma_{s_j}}{\bar{x}_j}$. We consider a general distribution for the inter-arrival time as well as the service time for each LC queue in the queuing model. Therefore, each LC queue can be modeled as a GI/GI/1 queue. The GI/GI/1 queue is referred to a single-server queue with first-in-first-out discipline and with a general distribution of the sequences for inter-arrival and service time (*GI* stands for general independent or general in brief).

The approximated expected response time of LC queue $j$ is computed by the following equation [34]:

$$E[\tilde{T}_j] = \bar{x}_j + \frac{C_{I_j}^2 - C_{S_j}^2}{2((\bar{x}_j)^{-1} - \tilde{\lambda}_j)} \tag{5}$$

In Eq. (5) we can replace $C_{I_j}^2$ by the following formula:

$$C_{I_j}^2 = \sigma_{I_j}^2 \tilde{\lambda}_j^2 = 1 + \tilde{P}_j\left((\hat{\lambda}_i)^2 \sigma_{I_i}^2 - 1\right) \tag{6}$$

Since $\tilde{P}_j = \frac{\tilde{\lambda}_j}{\hat{\lambda}_i}$, hence

$$C_{I_j}^2 = 1 + \frac{\tilde{\lambda}_j}{\hat{\lambda}_i}\left((\hat{\lambda}_i)^2 \sigma_{I_i}^2 - 1\right) \tag{7}$$

The objective function expresses in Eq. (3) is solved by an extended version of the approach developed by Li [35] (see Appendix), so the optimal arrival rate for LC queue $j$ will be

$$\tilde{\lambda}_j = \frac{1}{\bar{x}_j} - \frac{1}{\bar{x}_j}\sqrt{\frac{1 - (\hat{\lambda}_i)^2 \sigma_{I_i}^2 + (C_{S_j}^2 - 1)\hat{\lambda}_i \bar{x}_j}{1 - (\hat{\lambda}_i)^2 \sigma_{I_i}^2 + 2\hat{\lambda}_i(\bar{x}_j - z)}} \tag{8}$$

If we use the constraint $C(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i}) = \tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_{N_i} - \hat{\lambda}_i = 0$, we have

$$-\hat{\lambda}_i + \sum_{j=1}^{N_i} \frac{1}{\bar{x}_j} = \sum_{j=1}^{N_i} \frac{1}{\bar{x}_j}\sqrt{\frac{1 - (\hat{\lambda}_i)^2 \sigma_{I_i}^2 + (C_{S_j}^2 - 1)\hat{\lambda}_i \bar{x}_j}{1 - (\hat{\lambda}_i)^2 \sigma_{I_i}^2 + 2\hat{\lambda}_i(\bar{x}_j - z)}} \tag{9}$$

A closed form solution for Eq. (9) is impossible [35]. Therefore, we use a numerical solution proposed by Li [35]. A numerical solution uses bisection algorithm and searches $z$ in a range of $[lb, ub]$. From Eq. (8). $\tilde{\lambda}_j \geq 0$ we can get

$$z \leq \bar{x}_j - \frac{(C_{S_j}^2 - 1)\bar{x}_j}{2} \tag{10}$$

For all $1 \leq j \leq N_i$, therefore the upper bound of $z$ is

$$ub = \min\left(\bar{x}_j - \frac{(C_{S_j}^2 - 1)\bar{x}_j}{2}\right) \tag{11}$$

And in Eq. (9) let

$$\phi_j(z) = \frac{1}{x_j}\sqrt{\frac{1 - (\hat{\lambda}_i)^2 \sigma_{I_i}^2 + (C_{S_j}^2 - 1)\hat{\lambda}_i \bar{x}_j}{1 - (\hat{\lambda}_i)^2 \sigma_{I_i}^2 + 2\hat{\lambda}_i(\bar{x}_j - z)}} \tag{12}$$

Therefore, we simplify the Eq. (9) as follows:

$$\sum_{j=1}^{N_i} \frac{1}{\bar{x}_j} - \hat{\lambda}_i = \sum_{j=1}^{N_i} \phi_j(z) \tag{13}$$

If we consider Eq. (13), we have

$$\sum_{j=1}^{N_i} \phi_j(ub) \geq \sum_{j=1}^{N_i} \frac{1}{\bar{x}_j} - \hat{\lambda}_i \tag{14}$$

And lower bound of $z$ can be worked out based on Eq. (15). $lb$ can be obtained by dividing by 2 $ub$ repeatedly until the following condition is met:

$$\sum_{j=1}^{N_i} \phi_j(lb) \leq \sum_{j=1}^{N_i} \frac{1}{\bar{x}_j} - \hat{\lambda}_i \tag{15}$$

## 5 Proposed load balancing policy

The proposed load balancing policy in each reporting node is comprised of two parts. The first part determines how the analysis mentioned in the previous section is applied in each reporting node for job allocation to each logical cluster. In fact, the first part finds optimal arrival rate $\tilde{\lambda}_j$ for each logical cluster. The second part concerns the dispatch policies in each reporting node among logical clusters based on the routing probability gained by the first part.

### 5.1 Job allocation policy

In the analysis of Sect. 4, we consider several assumptions as follows:

- Each logical cluster in every reporting node have a GI/GI/1 queue;
- each logical cluster queue serves arriving requests in the FCFS fashion;
- to serve a request, its resource is found by the round robin policy within target logical cluster a way that this resource satisfies the QoS constraints of the request;
- a request (job) has bag of task (BoT) structure.

We use general distribution for the service time of each logical cluster. The average service time for each request in the logical cluster $j$ can be approximated as follows:

$$\bar{x}_j = \bar{S}_{\text{BoT}} \bar{E}_{\text{BoT}} \frac{S_m}{\bar{S}_j} \quad 1 \leq j \leq N_i \tag{16}$$

In Eq. (16) the service time of each request can be approximated as the product of the average BoT size and the average execution time of each task. This value is scaled by the division of maximum average processing speed of all logical clusters in each reporting node by average processing speed of cluster $j$. The job allocation policy to logical clusters in each reporting node is represented in the form of pseudo-code in Algorithm 1.

In Algorithm 1, the average service time of logical cluster $j$ is computed based on Eq. (16) in Steps 1 to 3. In Step 4 $ub$ is calculated based on Eq. (11) as the minimum value among all computed values for logical clusters in the previous steps. $lb$ is initialized with half of $ub$ at Step 5 and halved until the condition in Step 6 based on Eq. (15) is satisfied. Steps 10–15 show the bisection algorithm mentioned in previous section to find the proper value of $z$. $\varepsilon$ is the expected precision at Step 10 and initializes to $10^{-7}$. Finally, in Steps 17–19 the optimal arrival rate for each logical cluster is determined.

**Algorithm 1:** Job allocation policy to logical clusters in each reporting node

Input: $\hat{\lambda}_i, \sigma^2_{I_i}, \bar{x}_j$ for all $1 \leq j \leq N_i$

Output: $\tilde{\lambda}_j$ the optimal arrival rate of requests to different logical clusters, for all $1 \leq j \leq N_i$

**for** $j \leftarrow 1$ **to** $N_i$ **do**
  $\bar{x}_j = \bar{S}_{\text{BoT}} \bar{E}_{\text{BoT}} \frac{S_m}{S_j}$
**end**
$ub \leftarrow \min(\bar{x}_j - \frac{(C^2_{S_j}-1)\bar{x}_j}{2})$ for all $1 \leq j \leq N_i$
$lb \leftarrow \frac{ub}{2}$
**while** $\sum_{j=1}^{N_i} \phi_j(lb) > \sum_{j=1}^{N_i} \frac{1}{\bar{x}_j} - \hat{\lambda}_i$ **do**
  $lb \leftarrow \frac{lb}{2}$
**end**
Find the Lagrange multiplier $z$ to solve Eq. (9) by searching $z$ between the range $[lb, ub]$ using the bisection algorithm.
**while** $(ub - lb > \varepsilon)$ **do**//$\varepsilon$ is the expected precision
  $z \leftarrow (\frac{lb+ub}{2})$
  **if** $(\sum_{j=1}^{N_i} \phi_j(z) \leq \sum_{j=1}^{N_i} \frac{1}{\bar{x}_j} - \hat{\lambda}_i)$ **then**
    $z \rightarrow ub$
  **else** $z \rightarrow lb$
**end**
Compute the optimal arrival rate by Eq. (8) for each logical cluster
**for** $j \leftarrow 1$ **to** $N_i$ **do**
  $\tilde{\lambda}_j \leftarrow \frac{1}{\bar{x}_j} - \frac{1}{\bar{x}_j} \sqrt{\frac{1-\hat{\lambda}_i^2\sigma^2_{I_i} + (C^2_{S_j}-1)\hat{\lambda}_i\bar{x}_j}{1-\hat{\lambda}_i^2\sigma^2_{I_i} + 2\hat{\lambda}_i(\bar{x}_j - z)}}$
**end**

### 5.2 Dispatch policies

We consider three dispatch policies among logical clusters, namely BilRCDP, BerRCDP, and NRCDP. These policies differ in two ways. First, they differ in how they compute routing probabilities $\tilde{P}_j (\tilde{P}_j = \frac{\tilde{\lambda}_j}{\hat{\lambda}_i})$ for each logical cluster. Second, they differ in how they choose the sequence of requests sent to each logical cluster. BilRCDP and BerRCDP use the optimal arrival rate that is computed by the proposed load balancing policy; whereas, NRCDP is used just for comparison with these two policies. In fact, NRCDP is the simplest way for brokering in this case. In this dispatch policy, the routing probability $\tilde{P}_j$ for any logical cluster is considered to be equal, and logical clusters are sorted according to their average CPU speed. Within each logical cluster, resources are examined for QoS constraints of request in a round robin manner.

Both BerRCDP and BilRCDP use the same routing probabilities. In these two policies, the arrival rate for each logical cluster $(\tilde{\lambda}_j)$ is computed by Algorithm 1. The routing probability of these polices for each logical cluster in reporting node $i$

**Algorithm 2:** BilRCDP dispatch policy at each reporting node

Input: $\tilde{\lambda}_j, \bar{x}_j$ for all logical cluster $j$, $1 \leq j \leq N_i$
Output: selected resource ($r_j$)

1. *fastestLCluster* ← *findFastestLogicalCluster*($\bar{x}$)
2. **foreach** *logical cluster j* do
3. $X_j \leftarrow 0$
4. $Y_j \leftarrow 0$
5. **end**
6. $X_{fastestLCluster} \leftarrow 1$
7. min ← max *Value*
8. **foreach** *logical cluster j* **do**
9. $C = \frac{X_j + Y_j}{\tilde{P}_j}$
10. **if** ($c <$ min) **then**
11. min ← $C$
12. *selCluster* ← $j$
13. **end**
14. **end**
15. $Y_{selCluster} \leftarrow Y_{selCluster} + 1$
16. **foreach** *resource m in SelCluster* **do**
17. **if** (*resource m satisfy QoS constraints*) **then**
18. $r_j \leftarrow resource_m$
19. **else** check another resource in *SelCluster* based on round robins manner
20. **end**
21. **if** (none of resources in *SelCluster does not satisfy QoS constraints)* **then**
22. goto 7
23. **else** return($r_j$)

can be computed as $\tilde{P}_j = \frac{\tilde{\lambda}_j}{\hat{\lambda}_i}$. These policies differ in how they choose the sequences of requests sent to each LC queue.

In the BerRCDP policy, routing probabilities are used without any special sequencing of requests sent to each LC queue. Thus, BerRCDP is memory-less that it does not consider which request is sent to which LC queue; whereas, the BilRCDP policy takes into account the past sequence of routing with a little overhead. BilRCDP is a generalized form of round robin manner, and it considers the sequence of routing called the *billiard* sequence [36]. The authors in [36] suggested the method to implement the billiard sequence, and they generated it as follows:

$$j_b = \min_{\forall j} \left\{ \frac{X_j + Y_j}{P_j} \right\} \tag{17}$$

where $j_b$ is a target queue, and $X_j$ and $Y_j$ are vectors of integers with size $n$. $Y_j$ keeps the number of requests sent to queue $j$ and $X_j$ specifies which queue is the fastest. $X_j$ is set to one for the fastest queue and zero for other queues [33]. $Y_j$ is initialized

**Table 3** Comparison among three dispatch policies

| Criteria | BilRCDP | BerRCDP | NRCDP |
|---|---|---|---|
| How to compute routing probability for each logical cluster | The routing probabilities of logical clusters are different and they are computed by Algorithm 1 | The routing probabilities of logical clusters are different and they are computed by Algorithm 1 | The routing probabilities of logical clusters are equal |
| The sequence of choosing target logical cluster | Billiard sequence | Random | Round-robin strategy |
| Time complexity of finding target logical cluster | $O(N_i)$ | $O(1)$ | $O(1)$ |
| Time complexity of searching within target logical cluster | $O(\frac{R_i}{N_i})$ | $O(\frac{R_i}{N_i})$ | $O(\frac{R_i}{N_i})$ |

to zero, and it is updated to $Y_{j_b} = Y_{j_b} + 1$ after selecting the target queue. $P_j$ is a routing probability of incoming requests that are sent to the queue $j$.

Algorithm 2 demonstrates BilRCDP dispatch policy. In this algorithm, initially, the fastest logical cluster is found based on average service time for each logical cluster in Step 1. $\tilde{P}_j$ is sent to this algorithm based on Algorithm 1, and $\bar{x}_j$ is computed by Eq. (16). $X, Y$ variables are initialized to zero in Steps 2 to 5. One is assigned to the fastest logical cluster in Step 6. $Y_j$ shows the number of requests that are dispatched to logical cluster $j$, and initially assigns to zero at Step 4. In Steps 8–14, the value of adapted billiard sequences are computed, and the logical cluster with minimum value is selected. Then, $Y_j$ of selected logical cluster is incremented by 1 at Step 15. In Steps 16 to 20, all resources within selected logical cluster are examined for QoS constraints. If one of the resources satisfies QoS constraints, this resource will be selected at Step 18; otherwise, other resources on this logical cluster will be examined by a round robin manner. The round robin policy within each logical cluster is justifiable, because resources that their CPU speed is close are grouped in the same logical cluster. If none of the resources satisfies the QoS constraints, another logical cluster will be selected and this process continues (Steps 21, 22).

The BerRCDP dispatch policy selects the random logical cluster based on routing probability computed by Algorithm 1. In this dispatch policy, selection of logical cluster is random, but each logical cluster gives a request based on its routing probability. After selecting a logical cluster, each resource in the target logical cluster is examined for QoS constraints in the round robin manner.

Table 3 gives a comparison among three dispatch policies. These policies are compared according to the method for computing the routing probability of each logical cluster, the sequence of choosing target logical cluster, and time complexity. The time complexity of each dispatch policy is computed in two cases. The first one is for finding target logical cluster. In this case, the time complexity of BilRCDP is higher than the others and it is related to number of logical clusters in the reporting node, whereas in the second case it is computed for finding suitable resource to serve a

request within selected logical cluster. The time complexity in this case is equal for all three policies as it is related to number of resources in one logical cluster.

## 6 Performance evaluation

In order to evaluate the performance of proposed policies, we implemented CycloidGrid simulator as a discrete event simulator. CycloidGrid is written in Java and it is an extended version of Cycloid simulator [30] to emulate the P2P-based volunteer computing systems.

Physical network in CycloidGrid is emulated by Brite topology generator [37]. A physical network with $n$ computers which are connected by Waxman model and different link bandwidth are generated by Brite topology generator. The bandwidth between two nodes is between 10 Mbps to 1 Gbps with uniform distribution [31, 44].

Xtremlab trace [38] is used in this research to emulate resources in the CycloidGrid simulator. Xtremlab trace is exported from BOINC database where the information is collected by client or server in the BOINC.

The coefficient of variance (CV = StDev/Mean) of the service time within each logical cluster can be assumed as 1.1 ($C_{S_j}$) to model the performance variability of resources in volunteer computing systems according to the Xtremlab traces.

The performance metric related to the response time of requests to be considered in all simulation scenarios is average response time (ART). The ART of R given requests is defined as follows:

$$\text{ART} = \frac{\sum_{j=1}^{R}(w_j + \sum_{k=1}^{l_j} d_{k_j})}{R} \tag{18}$$

where $w_j$ is the waiting time of request $j$, $l_j$ is the number of tasks in request $j$, and $d_{k_j}$ is the weighted run time of task $k$ in request $j$. The weighted run time of each task is a scaled down value of run time on a computer with higher speed. The waiting time of request $j$ is computed by the following equation:

$$w_j = L_{c2i} + \max(L_{i2r'}) + L_{i2r} + \text{Max}\left(L_{c2r,} \sum_{l} d_l\right) \tag{19}$$

where $L_{c2i}$ is the communication overhead for sending a request between the client node and the injection node, $L_{i2r'}$ represents the communication delay between the injection node and each of selected reporting nodes. The maximum of this time is added to the waiting time of the job, because the injection node contacts the selected reporting nodes in parallel. $L_{i2r}$ represents communication overhead between the injection node and the run node. Also, the last term is the maximum of communication delay between the client node and the run node ($L_{c2r}$) for sending input files and the summation of run time for waiting tasks in the run node's queue.

A network model based on queuing theory is applied to compute a communication overhead between two peers [20]. In this analytical model, each connection between two peers is modeled by a GI/GI/1 queue, and it is assumed that each peer receives two types of traffic: background traffic of the Internet and a part of workload traffic that is imported into the system by the client nodes. Therefore, we have compound

distribution with arrival rate $\lambda_m$ and variance $\sigma^2_{I_m}$ as input traffic to each peer. Thus, the communication overhead between two peers can be computed by the following equation [20]:

$$\bar{L} = \Psi_m \left[ 2 + \frac{\sigma_m^2 \lambda_m^2}{2(1 - \lambda_m \Psi_m)} \right] + \sum_{i=s+1}^{d} \Psi_{m_i} \tag{20}$$

where $\Psi_m$ is the service time of associated queue to each connection, and it is calculated as follows:

$$\Psi_m = 0.5\alpha_{net} + F\beta_{net} \tag{21}$$

where $\alpha_{net}$ is the network latency, $\beta_{net}$ is an inverse of bandwidth along the link between two adjacent peers based on routing algorithm in the P2P network, and $F$ is a flow size transmitted between two peers. The last term of Eq. (20) is calculated as a summation of $\Psi_m$ along the route between adjacent peer to the source node and destination peer based on routing algorithm of the P2P overlay network. Interested readers can refer to our previous work [20] for more detail.

### 6.1 Workload model

The workload model for simulations is based on Grid Workload Archive [39]. In this model, the inter-arrival time has Weibull distribution, and the request duration follows Normal distribution as listed in Table 4. As in the volunteer computing systems the task duration is large, the square of task duration is considered.

Each BoT request can have QoS constraints including minimum CPU speed, minimum RAM, disk space requirements, and the deadline. The methodology used by Irwin et al. [45] is utilized to assign the deadline to each request. According to this methodology, the requests are classified into two classes. These classes are Low Urgency (LU) jobs and High Urgency (HU) jobs. A BoT request in HU class has low ratio of deadline to runtime; whereas, a request in LU class has high ratio of deadline. In our experiments, the ratio of deadline for HU requests follows normal distribution with mean 4 and variance 2; meanwhile, the ratio of deadline for LU requests is three times longer with mean 12 and variance 6.

We generate the workload for 1 day, where 2.5 hours is considered as the warm-up phase to avoid bias before the system reaches steady-state. Each experiment is

**Table 4** Input parameters for the workload model

| Parameters | Distribution/Value | Reference |
|---|---|---|
| Inter-arrival time | Weibull ($5 \leq \alpha \leq 9$, $\beta = 4.25$) | [39] |
| No. of tasks | Weibull ($\alpha = 2.11$, $\beta = 1.76$) | [39] |
| Task duration | Normal ($2.73 \leq m \leq 8.5$, $\sigma = 6.1$) | [39] |
| Internet inter-arrival time | Weibull ($\alpha = 0.06$, $\beta = 0.15$) | [43] |
| Internet flow size | Pareto ($\alpha = 3$, $\beta = 1.05$) | [43] |
| Inter-arrival time of peer churn | Poisson ($0.66 \leq \tau \leq 4.83$) | [21] |

performed on each of these workloads separately. For the sake of accuracy, each experiment is carried out several times by using different workloads and average of results is reported. In all the reported results, CV is less than 0.01. The number of resources is equal to 1000 and 3000 peers with heterogeneous computing speeds.

In order to generate different workloads, we modified two parameters one at a time. Therefore, to change the inter-arrival time, we modified the first parameter of Weibull distribution (the scale parameter $\alpha$) as shown in Table 4. Thus, the number of jobs increases from 10000 (i.e. $\alpha = 9$) to 19000 (i.e. $\alpha = 5$). Also, to have requests with different duration, the mean of normal distribution changes from 2.73 to 8.5 that is mentioned in Table 4. The average task duration in BoT changes from *44* to *109* minutes.

Peer churn is modeled by a Poisson distribution [21] with average inter-arrival time ($\tau$) differentiates from 0.66 minutes to 4.83 minutes that is presented in Table 4. Thus, from 10 % to 70 % of peers leave the system when average inter-arrival time varies from 4.83 to 0.66 minutes; whereas, some nodes join the system.

We consider the background traffic of the Internet follows the Weibull distribution [43] as shown in Table 4. Also, the Internet flow size follows the Pareto distribution according to [43]. The mean of Pareto distribution is considered as the flow size for the Internet traffic.

Each BoT request is assumed to have an input file. A ratio of communication cost to computation cost is applied in various studies on scheduling BoT requests [40]. This ratio is called communication-to-computation ratio (CCR). Therefore, we consider the file size of each BoT request is CCR times of its computation time. It is worth noting that in this research, we focus on balanced BoT application in which computation and communication time are important. Thus, a BoT request with CCR = 2 is taken into account.

### 6.2 Simulation results

Figures 4, 5, 6, 7 present ART versus inter-arrival time and average task duration for different policies. In these figures, average task duration is kept in the medium size (66.55 minutes) for ART versus inter-arrival time. Also, the inter-arrival time has kept in the medium size (i.e. $\alpha = 7.86$) for ART versus average task duration. Each request has minimum CPU speed, minimum RAM or disk space requirements as QoS constraints.

In Fig. 4, we consider 1000 peers in the system where the system is relatively static, and no peer joins or leaves during the experiment. As we expected, by reduction of inter-arrival time, the ART dramatically increases too. But the BilRCDP policy controls the ART by distributing the load evenly in the system. Meanwhile, NRCDP approaches the saturation point exponentially. BilRCDP marginally surpasses the BerRCDP with improvement factor of 23 %, 19 % in Fig. 4(a) and Fig. 4(b), respectively. The improvement of BilRCDP in theses figures with respect to NRCDP is 36 % and 29 %, respectively. Figures 4(c) and 4(d) compares these policies with the CAN policy proposed by Kim et al. [21]. This study is selected for comparison because they considered minimum CPU speed, RAM, and disk space requirements as QoS constrains of request, and the load balancing policy is implemented in their

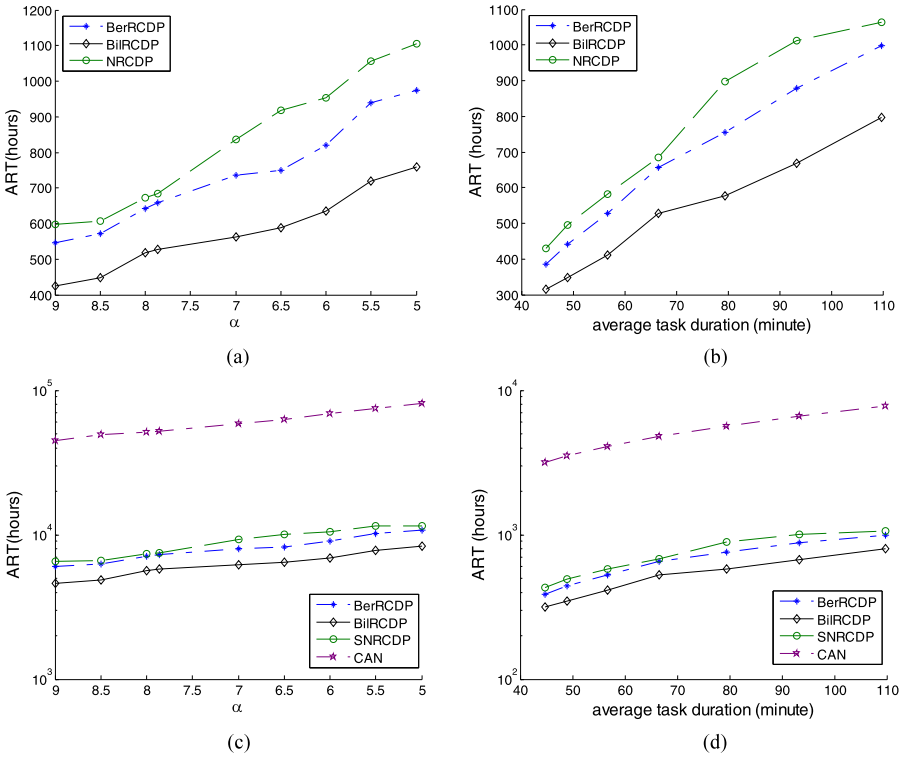


**Fig. 4** Average response time resulting from different policies with 1000 peers and static environment. The experiments are carried out by modifying (**a**, **c**) the $\alpha$ parameter in inter-arrival time, (**b**, **d**) the average duration of task in BoT

work. As one can see in Fig. 4(c) and 4(d), ART of the proposed policies is much lower than CAN. However, the overhead of CAN is lower than our proposed load balancing policies. The average number of messages sent for each request in CAN is almost 20 messages, whereas this average increases to 50 messages per request in our proposed policies. The number of messages in the proposed policies is 2.5 times of CAN; meanwhile, the improvement factor of BilRCDP is 70 % compare to CAN. The most of messages in the proposed policies are exchanged among the injection node and selected reporting nodes and a small fraction of them are sent for managing churn in the system.

In Fig. 5, the number of peers is the same as the previous experiment, but peers join or depart from the system with the average inter-arrival time $\tau = 2.38$ minutes. In this experiment, after 1000 nodes initially join the system, some nodes leave; meanwhile, some nodes join the system. The departure rate of peers in this experiment is 20 % of all peers in the system. In this experiment, when a node leaves the system, all of assigned job on the leaving node are reassigned to another peers. Because the leaving peers are selected randomly, possibly selecting the nodes with fewer jobs causes the reduction of ART in some situations such $\alpha = 6$ in Fig. 5(a) for BerRCDP and NRCDP policies. In this experiment BilRCDP surpasses BerRCDP and NRCDP

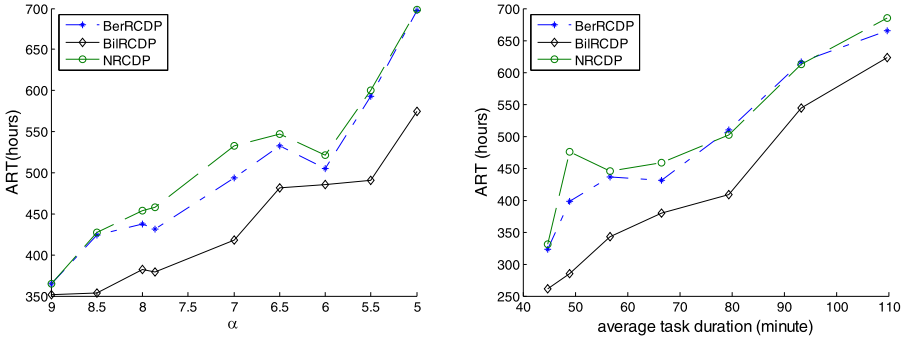


**Fig. 5** Average response time resulting from different policies with 1000 peers and dynamic environment. The experiments are carried out by modifying (**a**) the $\alpha$ parameter in inter-arrival time, (**b**) the average duration of task in BoT

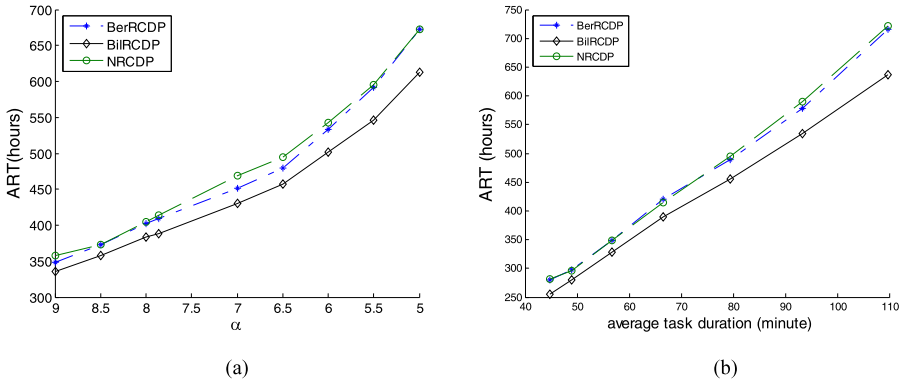


**Fig. 6** Average response time resulting from different policies with 3000 peers and static environment. The experiments are carried out by modifying (**a**) the $\alpha$ parameter in inter-arrival time, (**b**) the average duration of task in BoT

with the improvement factor of 18 %, 21 % in Fig. 5(a) and 17 %, 22 % in Fig. 5(b), respectively.

In Fig. 6, we increase the number of peers to 3000 peers in the system, but the system keeps in the static state and no node joins or leaves the system during the simulation. As it is shown, the BilRCDP still achieves a better performance with respect to BerRCDP and NRCDP with improvement factor of 8 %, 10 % in Fig. 6(a) and 7 %, 8 % in Fig. 6(b), respectively. The performance of BerRCDP decreases with increasing number of peers in the system. As we explained in Sect. 5, after selecting a resource by BerRCDP sequence, this resource is examined on QoS constraints. If it meets the QoS constraints, it will be selected; otherwise, another resource are examined. The QoS constraints have the performance impact on BerRCDP policy by changing the recommended sequence in this policy. This impact is less effective on the performance of the BilRCDP policy.

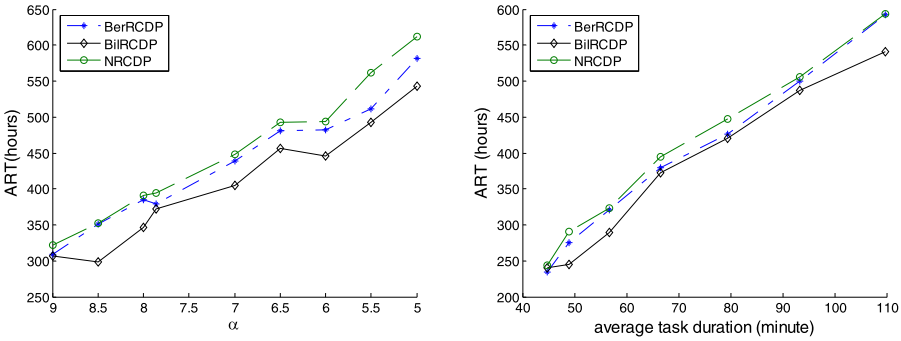


**Fig. 7** Average response time resulting from different policies with 3000 peers and dynamic environment. The experiments are carried out by modifying (**a**) the $\alpha$ parameter in inter-arrival time, (**b**) the average duration of task in BoT

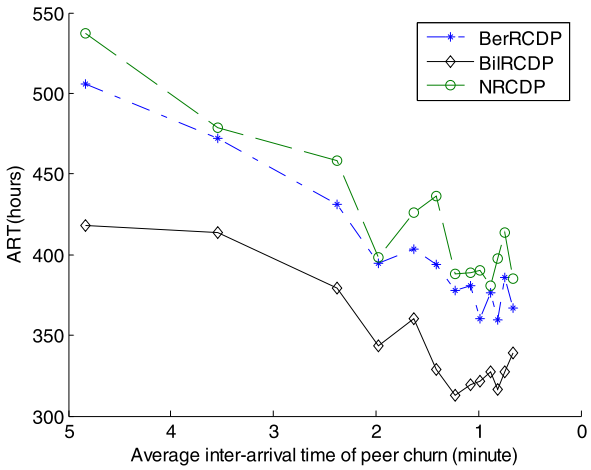


**Fig. 8** Average response time resulting from different policies with 1000 peers. The simulations are carried out by modifying the average inter-arrival time of peer churn

Figure 7 shows the experimental results for 3000 peers in the dynamic environment. In this experiment, peers leave or join to the system with average inter-arrival time $\tau = 2.38$ minutes. The departure rate of peers from the system is 20 % of peers similar to the second experiment. The BilRCDP has improvement factor of 8 %, 14 % in Fig. 7(a) and 5 %, 8 % in Fig. 7(b) with respect to BerRCDP and NRCDP, respectively.

Figure 8 presents ART for 1000 peers versus the average inter-arrival time of peer churn. The average inter-arrival time of peer churn varies from 4.83 minutes to 0.66 minutes. However, from 10 % ($\tau = 4.83$ minutes) to 70 % ($\tau = 0.66$ minutes) of all peers (with step of 5 %) leave the system; meanwhile, some nodes join the system. In this figure, the inter-arrival time and average task duration of BoT request are kept in the medium size ($\alpha = 7.86$, avg. task duration = 66.55 minutes).

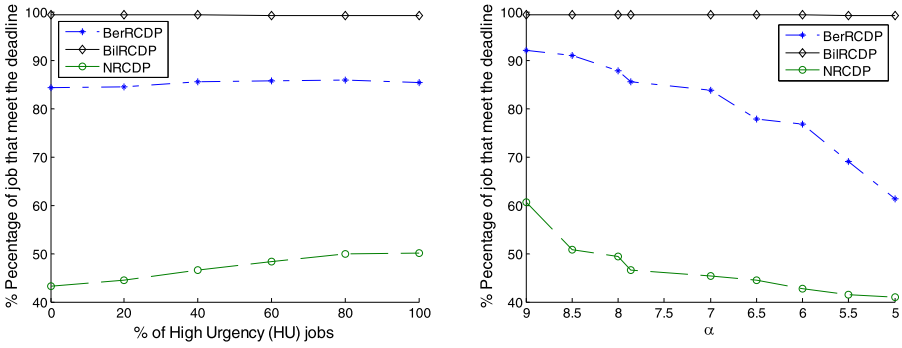


**Fig. 9** The percentage of jobs that meet the deadline versus the percentage of high urgency jobs (**a**) and the $\alpha$ parameter in inter-arrival time (**b**)

As illustrated in this figure, BilRCDP, BerRCDP, and NRCDP have similar behavior with decreasing the average inter-arrival time till 1.98 minutes. In this point, 25 % of peers leave the system. After that they start to oscillate, whereas changes of ART in NRCDP have a bigger step than the other two policies. This simulation shows that the system is resistant against the churn. The performance of the system does not decrease, if the churn rate increases.

Figure 9 presents the impact of high urgency jobs and arrival rate on the percentage of jobs that meet the deadline. In this experiment, the number of peers equals 1000 peers in static environment, and we consider the jobs with deadline, minimum CPU speed, and minimum RAM or disk space requirements as QoS constraints. In Fig. 9(a), various percentage of HU jobs are considered. For example, if the percentage of HU jobs is 40 %, the percentage of remaining LU jobs is 60 %. Also, the inter-arrival time and average task duration of BoT request are kept in the medium size ($\alpha = 7.86$, avg. task duration = 66.55 seconds). We decrease the average task duration in this workload from minute to second; therefore, the BoT execution time is decreased from hours to minutes. The reason is VC requests normally have long deadline in order of weeks considering availability of resource because of dynamic nature of volunteers [46]; thus, reduction of execution time can simulate the high urgency requests with meaningful deadlines in the VC environments. Meanwhile, in Fig. 9(b) the average task duration and the percentage of HU jobs are kept in the medium size (avg. task duration = 66.55 seconds, the percentage of HU jobs = 40 %). As depicted in Fig. 9, almost 99 % of jobs meet the deadline by BilRCDP policy, and this policy is robust with respect to increase of HU jobs and arrival rate of jobs. BerRCDP has almost stable behavior with increase of HU jobs while its performance decreases with increase of arrival rate. It shows that BilRCDP distributes the load more evenly than BerRCDP. Because the percentage of missed deadline jobs is increased in BerRCDP with increase of arrival rate. The performance of NRCDP is increased with the increase of high urgency jobs; meanwhile, its performance is decreased with increase of arrival rate.

## 7 Conclusions

In this paper, we propose an analytical model for load balancing in peer-to-peer based-volunteer computing systems. We consider the requests are arriving into the system in BoT form, and each request has some QoS constraints such as minimum CPU speed, minimum RAM or disk space requirements, and deadline. The proposed policies for load balancing are based on distributed parallel queues and knowledge-free approach; therefore, it does not impose any additional overhead on the system. The proposed resource discovery algorithm has two phases. In the first phase, it takes into account the load balancing feature in the system; whereas, in the second phase the proximity-aware feature is considered and the resource with minimum communication overhead is selected. Three dispatch policies, namely BilRCDP, BerRCDP, and NRCDP, are considered to distribute requests to the peers in the system. We compared the performance of these policies in different circumstances. The results of the experiments indicated that BilRCDP significantly decreases the average response time of the system with improvement factor of 13.12 %, 18.5 % in average with respect to BerRCDP and NRCDP, respectively. The proposed load balancing policy is proximity-aware, and selects the run node with minimum communication overhead; thus, it has better performance for BoT requests with large input/output file and considerable communication overhead. Also, the influence of the load balancing policy is highlighted for the longer running jobs, and high variations in job execution time.

As part of future work, we intend to consider data intensive application in the form of directed acyclic graph to evaluate the effectiveness of proposed policy on these applications. We think that proximity-aware feature can decrease the communication overhead on these applications sufficiently. Another interesting extension would be using Cloud resources in some of peers. Some applications have QoS requirements that could not be satisfied by the available resources of the VC systems, Clouds resources can be used in order to handle QoS requirements of these applications.

**Acknowledgements** This project was partially supported by Iran Telecommunication Research Centre (ITRC). The authors would like to thank Rodrigo N. Calheiros, Mohsen Amini, and Amir Vahid for useful discussions.

## Appendix: Proof of Eq. (8)

To solve the objective function of Eq. (3) we extend the approach developed by Li [35], since $\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_{N_i}$ is fixed, the problem is equivalent to minimize

$$Z(\tilde{\lambda}_1, \tilde{\lambda}_2, ..., \tilde{\lambda}_{N_i}) = \hat{\lambda}_i \hat{T}_i = \sum_{j=1}^{N_i} \tilde{\lambda}_j E(\tilde{T}_j) = \sum_{j=1}^{N_i} \left( \tilde{\lambda}_j \bar{x}_j + \frac{\tilde{\lambda}_j \bar{x}_j (C_{I_j}^2 - C_{S_j}^2)}{2(1 - \tilde{\lambda}_j \bar{x}_j)} \right)$$

By substituting Eq. (7) in the above equation, we have

$$Z(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i}) = \sum_{j=1}^{N_i} \left( \tilde{\lambda}_j \bar{x}_j + \frac{\tilde{\lambda}_j \bar{x}_j (1 + \frac{\tilde{\lambda}_j}{\hat{\lambda}_i}((\hat{\lambda}_i)^2 \sigma_{I_i}^2 - 1) - C_{S_j}^2)}{2(1 - \tilde{\lambda}_j \bar{x}_j)} \right)$$

In order to minimize $Z(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i})$ the Lagrange multiplier system is used,

$$\Delta Z(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i}) = z\Delta C(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i})$$

That is, we have $N$ equations

$$\frac{\partial Z(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i})}{\partial \tilde{\lambda}_j} = z\frac{\partial C(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i})}{\partial \tilde{\lambda}_j}$$

For all $1 \leq j \leq N_i$, and $C(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i})$ is the constraint $\tilde{\lambda}_1 + \tilde{\lambda}_2 + \cdots + \tilde{\lambda}_{N_i} - \hat{\lambda}_i = 0$, and $z$ is the Lagrange multiplier. We have

$$\frac{\partial Z(\tilde{\lambda}_1, \tilde{\lambda}_2, \ldots, \tilde{\lambda}_{N_i})}{\partial \tilde{\lambda}_j} = \frac{\bar{x}_j((\hat{\lambda}_i)^2\tilde{\lambda}_j\sigma_{I_i}^2(2-\tilde{\lambda}_j\bar{x}_j)+\tilde{\lambda}_j(-2+\tilde{\lambda}_j\bar{x}_j)-\hat{\lambda}_i(-3+C_{S_j}^2+4\tilde{\lambda}_j\bar{x}_j-2(\tilde{\lambda}_j)^2(\bar{x}_j)^2))}{2\hat{\lambda}_i(-1+\tilde{\lambda}_j\bar{x}_j)^2}$$

Thus, we can get $a_j(\tilde{\lambda}_j)^2 + b_j\tilde{\lambda}_j + c_j = 0$, where

$$a_j = -(\bar{x}_j)^2(\hat{\lambda}_i)^2\sigma_{I_i}^2 + (\bar{x}_j)^2 + 2(\bar{x}_j)^3\hat{\lambda}_i - 2\hat{\lambda}_i(\bar{x}_j)^2 z$$
$$b_j = 2\bar{x}_j(\hat{\lambda}_i)^2\sigma_{I_i}^2 - 2\bar{x}_j - 4\hat{\lambda}_i(\bar{x}_j)^2 + 4\hat{\lambda}_i\bar{x}_j z$$

and

$$c_j = 3\hat{\lambda}_i\bar{x}_j - \hat{\lambda}_i\bar{x}_j C_{S_j}^2 - 2\hat{\lambda}_i z$$

We have

$$\tilde{\lambda}_j = \frac{1}{\bar{x}_j} - \frac{1}{\bar{x}_j}\sqrt{\frac{1-(\hat{\lambda}_i)^2\sigma_{I_i}^2 + (C_{S_j}^2-1)\hat{\lambda}_i\bar{x}_j}{1-(\hat{\lambda}_i)^2\sigma_{I_i}^2 + 2\hat{\lambda}_i(\bar{x}_j - z)}}$$

We have steady state situation if $\tilde{\lambda}_j\bar{x}_j < 1 \Rightarrow \tilde{\lambda}_j < \frac{1}{\bar{x}_j}$, consequently the other value for $\tilde{\lambda}_j$ is not acceptable.

## References


1. Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D (2002) SETI@home: an experiment in public-resource computing. Commun ACM 45:56–61. doi:10.1145/581571.581573
2. Beberg AL, Ensign DL, Jayachandran G, Khaliq S, Pande VS (2009) Folding@home: lessons from eight years of volunteer distributed computing. In: Proceedings of IEEE international symposium on parallel and distributed processing (IPDPS), pp 1–8. doi:10.1109/IPDPS.2009.5160922
3. Fedak G, He H, Lodygensky O et al (2008) EDGeS: a bridge between desktop grids and service grids. In: Proceedings of the third ChinaGrid annual conference (ChinaGrid), pp 3–9. doi:10.1109/ChinaGrid.2008.44
4. Guinnessy P (2003) Climate@home. Phys Today 56:38. doi:10.1063/1.1650221
5. Anderson DP (2004) BOINC: a system for public-resource computing and storage. In: Proceeding of grid computing (Grid), pp 4–10. doi:10.1109/GRID.2004.14
6. Epema DHJ, Livny M, Dantzig RV, Evers X, Pruyne J (1996) A worldwide flock of condors: load sharing among workstation clusters. Future Gener Comput Syst 12:53–65
7. Litzkow MJ, Livny M, Mutka MW (1998) Condor—a hunter of idle workstations. In: Proceedings of international conference on distributed computing systems (ICDCS), pp 104–111. doi:10.1109/DCS.1988.12507
8. Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the condor experience. Concurr Pract Exp 17:323–356

9. Chien A, Calder B, Elbert S, Bhatia K (2003) Entropia: architecture and performance of an enterprise desktop grid system. J Parallel Distrib Comput 63:597–610
10. Cappello F, Djilali S, Fedak G, Herault T, Magniette F, Neri V, Lodygensky O (2005) Computing on large scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. Future Gener Comput Syst 21:417–437. doi:10.1016/j.future.2004.04.011
11. Chu X, Nadiminti K, Jin C, Venugopal S, Buyya R (2007) Aneka: next-generation enterprise grid platform for e-science and e-business applications. In: Proceedings of the IEEE international conference on e-science and grid computing (e-science), pp 151–159. doi:10.1109/E-SCIENCE.2007.12
12. Marosi AC, Gombas G, Balaton Z, Kacsuk P, Kiss T (2008) SZTAKI desktop grid: building a scalable, secure platform for desktop grid computing. In: Making grids work VII, pp 365–376. doi:10.1007/978-0-387-78448-9_29
13. Vladoiu M, Constantinescu Z (2009) Development journey of QADPZ—a desktop grid computing platform. Int J Comput Commun Control 4:82–91
14. Wolinsky DI, Agrawal A, Boykin P, Davis J, Ganguly A, Paramygin V, Sheng P, Figueiredo R (2006) On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations. In: Proceedings of international workshop on virtualization technology in distributed computing (VTDC), p 8. doi:10.1109/VTDC.2006.8
15. Abbes H, Cerin C, Jemni M (2008) PastryGrid: decentralisation of the execution of distributed applications in desktop grid. In: Proceedings of international workshop on middleware for grid computing (MGC), pp 1–6. doi:101145/1462704.1462708
16. Abbes H, Cerin C, Jemni M (2009) Bonjourgrid: orchestration of multi-instances of grid middlewares on institutional desktop grids. In: IEEE international symposium on parallel and distributed processing (IPDPS), pp 1–8. doi:10.1109/IPDPS.2009.5161140
17. Anglano C, Canonico M, Guazzone M, Botta M, Rabellino S, Arena S, Girardi G (2008) Peer-to-peer desktop grids in the real world: the ShareGrid project. In: Proceedings of IEEE international symposium on cluster computing and the grid, pp 609–614 (CCGrid 2008). doi:10.1109/CCGRID.2008.23
18. Butt AR, Zhang R, Hu CY (2006) A self-organizing flock of condors. J Parallel Distrib Comput 66:145–161
19. Byun E, Kim H, Choi S, Lee S, Han YS, Gil JM, Jung SY (2008) Self-gridron: reliable, autonomous, and fully decentralized desktop grid computing system based on neural overlay network. In: Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA), pp 569–575
20. Ghafarian T, Deldari H, Javadi B, Yaghmaee MH, Buyya R (2012) CycloidGrid: A proximity-aware P2P-based resource discovery architecture in volunteer computing systems. Future Gener Comput Syst (in press). doi:10.1016/j.future.2012.08.010
21. Kim JS, Nam B, Keleher P, Marsh M, Bhattacharjee B, Sussman A (2008) Trade-offs in matchmaking job and balancing load for distributed desktop grids. Future Gener Comput Syst 24:415–424. doi:10.1016/j.future.2007.07.007
22. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S (2001) A scalable content addressable network. In: Proceedings of the conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM), pp 161–172. doi:10.1.1.140.3129
23. Abdullah T, Alima LO, Sokolov V, Calomme D, Bertels K (2009) Hybrid resource discovery mechanism in ad hoc grid using structured overlay. In: Proceedings of the international conference on architecture of computing systems. Lecture notes in computer science, vol 5455. Springer, Berlin, pp 108–119
24. Mastroianni C, Cozza P, Talia D, Kelley I, Taylor I (2009) A scalable super-peer approach for public scientific computation. Future Gener Comput Syst 25:213–223. doi:10.1016/j.future.2008.08.001
25. Lazaro D, Marques JM, Vilajosana X (2010) Flexible resource discovery for decentralized P2P and volunteer computing systems. In: Proceedings of workshops on enabling technologies: infrastructure for collaborative enterprises (WETICE), pp 235–240. doi:10.1109/WETICE.2010.44
26. Di S, Wang CL, Hu DH (2009) Gossip-based dynamic load balancing in a self-organized desktop grid. In: Proceedings of the 10th high-performance computing Asia, HPCAsia, pp 85–92. doi:10.1.1.160.260
27. Ganesh A, Kermarrec AM, Massoulie L (2003) Peer-to peer membership management for gossip-based protocols. IEEE Trans Comput 52:139–149. doi:10.1109/TC.2003.1176982
28. Di S, Wang CL (2010) Conflict-minimizing dynamic load balancing for P2P desktop grid. In: Proceeding of 11th IEEE/ACM international conference on grid computing (grid), pp 25–28. doi:10.1109/GRID.2010.5697946

29. Chatrapati K, Ujwala Rekha J, Vinaya Babu A (2010) Competitive equilibrium approach for load balancing a computational grid with communication delays. J Theor Appl Inf Technol 19:126–133
30. Shen H, Xu C, Chen G (2006) Cycloid: a scalable constant-degree p2p overlay network. Perform Eval 63:195–216. doi:10.1016/j.peva.2005.01.004
31. Bouguerra MS, Kondo D, Trystram D (2011) On the scheduling of checkpoints in desktop grids. In: Proceeding of 11th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid), pp 305–313. doi:10.1109/CCGrid.2011.63
32. Ross SM (1997) Stochastic processes. Wiley, New York
33. Anselmi J, Gaujal B (2010) Optimal routing in parallel, non-observable queues and the price of anarchy revisited. In: Proceedings of 22th international tele traffic congress (ITC), pp 1–8. doi:10.1109/ITC.2010.5608745
34. Guo X, Lu Y, Squillante MS (2004) Optimal probabilistic routing in distributed parallel queues. ACM SIGMETRICS Perform Eval Rev 32:53–54. doi:10.1145/1035334.1035355
35. Li K (2008) Optimal load distribution in non dedicated heterogeneous cluster and grid computing environments. J Syst Archit 54:111–123. doi:10.1016/j.sysarc.2007.04.003
36. Hordijk A, der Laan DV (2004) Periodic routing to parallel queues and billiard sequences. Math Methods Oper Res 59:173–192. doi:10.1007/s001860300322
37. Medina A, Lakhina A, Matta I, Byers J (2001) BRITE: an approach to universal topology generation. In: Proceedings of international symposium on modelling, analysis and simulation of computer and telecommunication systems (MASCOTS), pp 346–353. doi:10.1.1.94.2118
38. Malecot P, Kondo D, Fedak G (2006) XtremLab: a platform for observation and characterization grids of PCs on the Internet. In: Proceeding of parallel meetings of the French (RenPar)
39. Iosup A, Sonmez O, Anoep S, Epema D (2008) The performance of bags-of-tasks in large-scale distributed systems. In: Proceedings of the international symposium on high performance distributed computing (HPDC), pp 97–108. doi:10.1145/1383422.1383435
40. da Silva FAB, Senger H (2011) Scalability limits of bag-of-tasks applications running on hierarchical platforms. J Parallel Distrib Comput 71:788–801
41. Kondo D, Anderson DP, McLeod J VII (2007) Performance evaluation of scheduling policies for volunteer computing. In: Proceedings of IEEE international conference on e-science and grid computing (e-science), pp 415–422. doi:10.1109/E-SCIENCE.2007.57
42. Anderson DP (2011) Emulating volunteer computing scheduling policies. In: Proceeding of IEEE international parallel & distributed processing symposium (IPDPS), pp 1839–1846. doi:10.1109/IPDPS.2011.343
43. Basher N, Mahanti A, Williamson C, Arlitt M (2008) A comparative analysis of web and peer-to-peer traffic. In: Proceeding of international world wide web conference (WWW), pp 287–296. doi:10.1145/1367497.1367537
44. Elwaer A, Harrison A, Kelley I, Taylor I (2011) Attic: A case study for distributing data in BOINC projects. In: IEEE international parallel & distributed processing symposium (IPDPS), pp 1863–1870. doi:10.1109/IPDPS.2011.348
45. Irwin D, Grit L, Chase J (2004) Balancing risk and reward in a market-based task service. In: IEEE international symposium on high performance distributed computing (HPDC), pp 160–169. doi:10.1109/HPDC.2004.1323519
46. Heien EM, Anderson DP, Hagihara K (2009) Computing low latency batches with unreliable workers in volunteer computing environments. J Grid Comput 7:501–518. doi:10.1007/s10723-009-9131-6